User-Level Communication in Cluster-Based Servers *

Enrique V. Carrera, Srinath Rao, Liviu Iftode, and Ricardo Bianchini
Department of Computer Science
Rutgers University
Piscataway, NJ 08854-8019

{vinicio,rao,iftode,ricardob}@cs.rutgers.edu

Technical Report DCS–TR–453, August 2001

Abstract

Clusters of commodity computers are currently being used to provide the scalability required by several popular Internet services. In this paper we evaluate an efficient cluster-based WWW server, as a function of the characteristics of the intra-cluster communication architecture. More specifically, we evaluate the impact of processor overhead, network bandwidth, remote memory writes, and zero-copy data transfers on the performance of our server. Our experimental results with an 8-node cluster and four real WWW traces show that network bandwidth affects the performance of our server by only 6%. In contrast, user-level communication can improve performance by as much as 29%. Low processor overhead, remote memory writes, and zero-copy all make small contributions towards this overall gain. To be able to extrapolate from our experimental results, we use an analytical model to assess the performance of our server under different workload characteristics, different numbers of cluster nodes, and higher performance systems. Our modeling results show that higher gains (of up to 55%) can be accrued for workloads with large working sets and next-generation servers running on large clusters.

1 Introduction

The number of Internet users has increased rapidly over the last few years. This large user base is placing significant stress on the computing resources of popular services available on the Internet. Clusters of commodity computers are currently being used to provide the scalability required by several of these services.

Several researchers and companies have concerned themselves with the cluster-based servers, e.g. [29, 31, 30, 24, 2, 3, 37, 11, 5, 12]. There are two main classes of servers in terms of how the clients' requests are distributed across the cluster: content-oblivious and content-aware servers. In a content-oblivious server the request distribution is based solely on a load metric, which is usually the number of open connections being handled by each node. Early servers were mostly of this type. However, content-aware servers are becoming ever more popular, as they can perform a more sophisticated request distribution. More specifically, in a content-aware server, request distribution decisions are affected by the actual content requested, so the cluster node that initially accepts a request (establishing a TCP connection with the client) and determines the content being requested may not be the node that should actually service the request. In that case, the request has to be forwarded to the most appropriate node, generating intra-cluster communication.

Content-aware distribution can be exploited to improve cache locality, as originally proposed in [24]. The idea is to aggregate the set of memories of the cluster into a large cache and distribute requests for files based on cache locality, as well as load balancing considerations. For their focus on cache locality, we refer to these servers as locality-conscious servers. We previously proposed and evaluated a portable locality-conscious WWW server, called PRESS, that relies heavily on efficient intra-cluster communication for good performance [12]. In fact, PRESS uses the Virtual Interface Architecture (VIA) [13] industry standard for user-level communication. User-level communication reduces the cost (or *processor overhead*) of message transfers by eliminating kernel traps and unnecessary data copying from the critical communication path. An interesting aspect of the standard is that it specifies remote memory accesses, i.e. operations that can read/write data from/to the correct memory addresses without intervention by the remote processor.

The evaluation in [12] showed that PRESS pays a small performance penalty for its portability, but did not consider the impact of the properties of the intra-cluster communication on the performance of our server. In particular, we did not consider the gains achievable by remote memory writes and zero-copy transfers. Furthermore, the effect of high overhead protocols and low bandwidth networks was only evaluated analytically for idealized workloads. Finally, the experimental evaluation was performed under favorable intra-cluster communication conditions, as the server nodes communicated over a gigabit-per-second network (Giganet) that implements VIA in hardware/firmware.

This paper extends our previous work by investigating the effect of the performance and features of the intra-cluster communication on the performance of PRESS. To set the study in context, we assessed the percentage of time each CPU running our server spends on intra-cluster communication, as opposed to external communication with clients and actually servicing requests. To determine this percentage we ran our server on an 8-node cluster using TCP and a Fast Ethernet switch for communication. Figure 1 presents the result of experiments with four WWW traces. The figure shows that more than 50% of the time is spent with intra-cluster communication for all traces, so there is great potential for user-level communication to

*This research has been supported by NSF under grant # CCR-9986046.

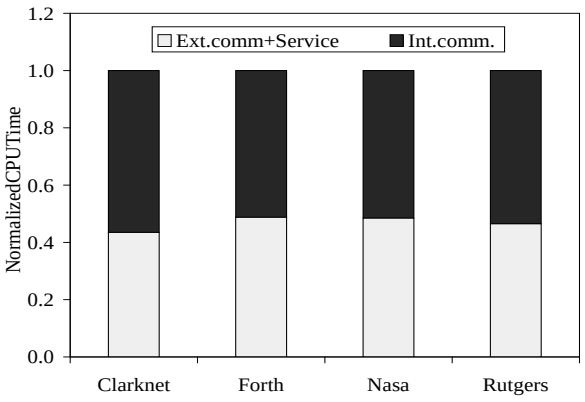

Figure 1: Time spent by PRESS (TCP/FE).

improve performance. (The details about the infrastructure behind these experiments are presented later.)

Thus, our three main goals are: (1) to assess the usefulness of remote memory writes and zero-copy in terms of performance; (2) to determine experimentally the intra-cluster communication parameters that have the greatest performance impact; and (3) to extrapolate our experimental results to investigate areas of the parameter space where user-level communication can be more useful. We accomplish these goals by evaluating versions of PRESS that utilize different protocol/network combinations and that exploit remote memory writes and zero-copy transfers to different extents. To be able to extrapolate from our experimental results, we use an analytical model of the performance of our server for varying communication characteristics, workload characteristics, and numbers of cluster nodes.

Our experimental results with an 8-node cluster and four real WWW traces show that network bandwidth affects the performance of PRESS by only 6%. In contrast, user-level communication can improve performance by as much as 29%. Low processor overhead, remote memory writes, and zero-copy all make small contributions towards this overall gain. (Network latency has no effect on throughput, so we did not consider it.) Our modeling results show that throughput gains of up to 49% can be accrued by user-level communication for large workload working sets and large numbers of nodes. These results also show that gains can reach 55% with next-generation servers, large working sets, and large numbers of nodes.

We believe that our findings have implications beyond our server. Communication in PRESS and other portable content-aware servers is so intensive that it is unlikely that other servers could accrue more significant improvements from user-level communication. Furthermore, even though we focus on servers with a content-aware request distribution and PRESS in particular, our observations should directly extend to other types (such as ftp, email, proxy, or file) and implementations of cluster-based servers, as long as files or file blocks are effectively transferred among the cluster nodes. A few examples of other servers in this class are the Swala WWW server [21], the Porcupine email server [30], and servers based on either the Federated File System [19] or Cooperative Caching Middleware [14].

The remainder of this paper is organized as follows. The next section section discusses user-level communication, VIA, PRESS, and the related work. Section 3 presents the methodology used in our experiments and discusses the performance results of our WWW server. Section 4 presents our model and its results. Finally, section 5 summarizes our findings and concludes the paper.

2 Background

2.1 User-Level Communication and VIA

The goal of user-level communication is to reduce the software overhead involved in sending and receiving messages, by removing the operating system from the critical communication path in a protected fashion. This goal is achieved by giving processes direct access to their network interfaces; the operating system in only involved in setting up the communication. In this way, the communicating parties can avoid intermediate copies of data, interrupts, and context switches in the critical path. The result of these optimizations is communication latency and bandwidth that approach the hardware limits.

There has been extensive research in user-level communication, e.g. [35, 17, 27, 34, 7, 6, 36, 1, 16]. This research led to the proposal of the VIA [15, 13] industry standard by a group of leading computer companies. In VIA, each communicating process can open directly accessible interfaces to the network hardware. Each interface, called *Virtual Interface* (VI), represents a communication end-point analogous to the socket end-point in a traditional TCP connection. In this way, pairs of VIs can be connected to create communication channels for bidirectional point-to-point data transfers. Each VI has a send and a receive queue. Processes post requests to these queues to send or receive data. The requests have the form of descriptors. Each descriptor contains all the information that the network interface controller needs to process the corresponding request, including pointers to data buffers. The network interface controller asynchronously processes the posted descriptors and marks them when completed. The processes then remove completed descriptors from the queues and reuse them for subsequent requests. In order to facilitate the removal of completed descriptors, VIA also specifies *Completion Queues* (CQs). A CQ can combine the notification of descriptor completions of multiple VIs into a single queue.

VIA provides two styles of communication. Besides the traditional send/receive data transfer model, it allows for direct access to the memory of remote machines. A remote memory access reads/writes data from/to the correct remote addresses without intervention by the remote processor. Regardless of the style of communication, the memory used for every data transfer in VIA needs to be "registered". The registration locks the appropriate pages in physical memory, allowing for direct DMA operations in the user memory buffers without the possibility of an intervening page replacement.

Finally, the VIA standard specifies three levels of reliability: unreliable delivery, reliable delivery, and reliable reception. In unreliable delivery, both regular and remote memory messages can be lost without being detected or retransmitted. In reliable delivery, all data submitted to transfer are guaranteed to arrive at the destination network interface exactly once and in order, in the absence of errors. If an error occurs, it is reported. Reliable reception is just like reliable delivery, except that a transfer is only successfully completed when the data have been delivered to the target memory location.

VIA has several successful implementations, e.g. Myrinet-VI [10], Giganet VIA [18], M-VIA [8], ServerNet VIA [32], FirmVIA [4]. Our experiments are made with Giganet VIA. This implementation supports VIA in hardware using its proprietary cLAN network

interfaces connected through 2.5 Gbits/s full-duplex links. Giganet VIA does not support remote memory reads (only remote writes are supported) and reliable reception.

2.2 PRESS

Overview. PRESS is our portable cluster-based, locality-conscious WWW server [12]. Although WWW servers must service requests for both static and dynamic contents, in this paper we focus solely on PRESS as a server of static content (read-only files), since this type of content puts the most stress on the intra-cluster network.

Just like other locality-conscious servers [24, 3, 11, 5], PRESS is based on the observation that serving a request from any memory cache, even a remote cache, is substantially more efficient than serving it from disk, even a local disk. Essentially, the server distributes HTTP requests across the cluster nodes based on cache locality and load balancing considerations, so that the requested content is unlikely to be read from disk if there is a cached copy somewhere in the cluster.

PRESS assumes that HTTP requests are directed to the cluster using a standard method, such as Round-Robin DNS or a content-oblivious, load balancing front-end device. Thus, any node of the cluster can receive a client request and becomes the *initial node* for that request. When the request arrives at the initial node, the request is parsed and, based on its content, the node must decide whether to service the request itself or forward the request to another node. A request for a large file ($\geq$ 512 KBytes in our prototype) is always serviced locally by the initial node. In addition, the initial node is chosen as the service node, if this is the first time the file is requested or it already cached the requested file. If neither of these conditions holds, the least loaded node that is caching the requested content becomes a candidate for *service node*. This node is chosen as the service node either when it is not overloaded (i.e. its number of open connections is not larger than a user-defined threshold, T = 80 in our experiments) or when it is overloaded but so are the initial node and the least loaded node in the cluster. In this way, popular files end up replicated at multiple nodes for better load balancing.

A forwarded request is handled in a straightforward way by the service node. If the requested file is cached, the service node simply transfers it to the initial node. If the file is not cached, the service node reads the file from its local disk, caches it locally, and finally transfers it to the initial node. Upon receiving the file from the service node, the initial node sends it to the client. The initial node does not cache the file received from the service node to avoid excessive file replication across the cluster.

In order to be able to intelligently distribute the HTTP requests it receives, each node needs locality and load information about all the other nodes. Locality information takes the form of the names of the files that are brought into the caches, whereas load information is represented by the number of open connections handled by the nodes.

The dissemination of caching information is straightforward. Whenever a node either replaces or starts caching a file, it broadcasts that fact to the other nodes. Broadcasts of caching information are very infrequent in steady-state.

The dissemination of load information can be done in one of two ways: broadcasting and piggy-backing. When broadcasting, each node broadcasts its current load when the load is a certain number of connections (the load threshold) greater or smaller than the last broadcast value. When piggy-backing, the current load is appended to any message sent among the nodes. In this way, each message updates the load information of its sender on the receiver. Most of the experiments in this paper were performed with piggy-backing.

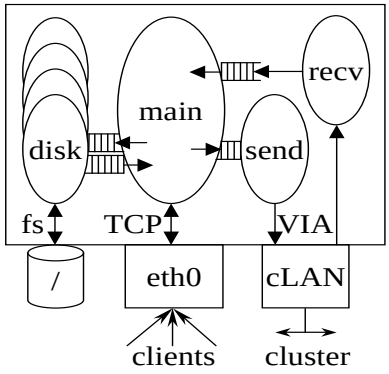

Figure 2: Basic PRESS architecture.

Communication architecture. PRESS is an event-driven server that was originally written to use VIA. Figure 2 shows a diagram with the basic architecture of our server. For highest performance, the main thread should not block. Thus, we use helper threads for disk access and communication. Disk threads are used to access files on disk to avoid stalling the main server thread, as suggested in [25].

In order to communicate inside the cluster, each cluster node sets up VI end-points with each other node. Two threads per node are responsible for sending and receiving messages; we refer to these threads as the send thread and the receive thread, respectively. These threads remain blocked until there is a message to be sent or received. The main server thread unblocks the send thread using a semaphore, after a *digest* of the message to be sent has been queued at a data structure that is shared by these two threads. When the send thread wakes up, it simply creates a message descriptor and places it at the corresponding send queue. The receive thread is unblocked by the arrival of a regular (as opposed to a remote memory write) message at one of the receive queues. When the receive thread wakes up, it simply determines which VI received the message and places a digest of the message at a data structure shared with the main thread. The main thread periodically polls this data structure. Remote memory writes have no effect on the receive thread. In fact, the receive thread is not even required, if all the communication is effected with remote memory writes.

Intra-cluster communication in PRESS occurs for five types of messages:

- Exchange of load information – very short messages carrying numbers of open connections;
- Exchange of caching information – short messages carrying file names;
- Request forwarding – short messages carrying file names;
- File transfer – long messages carrying file data; and
- Window-based flow control – very short messages carrying numbers of empty buffer slots.

As we explain above, load information exchanges may not require explicit messages because the local load values can be piggy-backed

in other messages. In section 3 we analyze different load dissemination strategies and experiment with several possible implementations for each of these message types. The implementations differ in terms of the extent to which they use remote memory writes and copying.

Implementations that use remote memory writes can improve performance by not involving the receiver in the transfer. However, remote memory writes can also require more messages if copies are to be avoided. Furthermore, remote memory writes can involve significant polling overhead, especially for cluster configurations with a large number of nodes, when messages cannot be overwritten and must be handled quickly.

As an example of this tradeoff, a request forwarding message has to be handled quickly to reduce latency, so the main thread has to poll several queues (one queue for each node) fairly frequently to determine whether another node needs to receive a file. The regular message implementation of this message type reduces the amount of polling to the main thread polling of the single data structure it shares with the receive thread. However, this latter implementation requires data copying from the message descriptor to the shared data structure, so that the descriptor can be re-utilized by another message as quickly as possible, reducing flow control stalls.

In contrast, no overhead is associated with remote memory writes for messages implementing flow control or carrying load information. These messages do not require immediate attention and can be overwritten, which makes their implementation with remote memory writes much more efficient than with regular messages.

Besides the VIA implementation of PRESS, we also have a prototype of our server that uses TCP. The TCP version basically has the same structure of its VIA counterpart; the main differences are the replacement of the VI end-points by TCP sockets and the elimination of flow control messages, which is implemented transparently to the server by TCP itself.

Performance. PRESS compares favorably against other servers in terms of performance. Its single-node throughput is equivalent to that of the Flash server [25], as these two servers are based on similar optimizations. Flash has been shown superior to Apache in [25]. In [12], we experimentally show that the multi-node throughput of the original version of PRESS on 8 nodes is within 7% of that of scalable LARD [3], a highly efficient but non-portable locality-conscious server. Using analytical modeling, we showed that portability should cost no more than 15% in terms of performance, even for large (96-node) clusters.

2.3 Related Work

The main contribution of this paper is to quantify the performance impact of user-level communication and several messaging characteristics on cluster-based servers. Previous studies of user-level communication were performed either for microbenchmarks [6, 35, 27, 36, 15, 10, 8, 32] or in the context of scientific applications [28, 23, 33]. This paper extends those studies to real, non-scientific applications, in particular to the now very popular cluster-based servers.

As far as we are aware, in only one other paper [22] has the performance of a server been evaluated as a function of network parameters. The study considered a single-processor NFS server and the effect of the parameters of the network that connects the server to its clients. In contrast, our work concentrates on the impact of network parameters and characteristics on the inter-node communication of cluster-based servers.

Logs	Num files	Avg file size	Num requests	Avg req size
Clarknet	28864	14.2 KB	2978121	9.7 KB
Forth	11931	19.3 KB	400335	8.8 KB
Nasa	9129	27.6 KB	3147684	21.8 KB
Rutgers	18370	27.3 KB	498646	19.0 KB

Table 1: Main characteristics of the WWW server traces.

The only previous communication sensitivity studies of cluster-based servers were done by ourselves in [5, 12]. However, our previous evaluations did not consider the ability to perform remote memory writes and to avoid data copies. Moreover, the effect of protocol overhead and network bandwidth was not assessed experimentally. PRESS and the analytical model we use in this paper were introduced in [12]. The server and model used here are modified versions of those in that paper. PRESS has been modified to use piggybacking, to communicate with TCP as well as VIA, and to exploit remote memory writes and zero-copy to various degrees. The model has been modified to consider remote memory writes and zero-copy.

Previous studies of VIA have been performed in the context of parallel and/or distributed applications [8, 32, 28]. Thus, this paper extends the small body of work on this industry standard, while evaluating several of its main features (low overhead, remote memory write, copy avoidance) for cluster-based servers. In contrast with this paper, other studies of remote memory write have concentrated on microbenchmarks (e.g. [17, 1]) and do not consider the impact of different messaging characteristics on the usefulness of this primitive.

3 Experimentation

3.1 Methodology

Cluster hardware and workloads. Our cluster is comprised by 8 Linux-based PCs with 300 MHz Pentium II processors, 512 KBytes of second-level cache, 512 MBytes of memory, a SCSI disk, and interfaces to switched Fast Ethernet and Giganet (cLAN) networks.

Besides our main cluster, we use another 10 Pentium-based machines to generate load for the servers. These clients communicate with the server using TCP over a Fast Ethernet switch. For simplicity, we did not experiment with a content-oblivious front-end device; we used the equivalent strategy of having clients send requests to the nodes of the server in randomized fashion with equal probabilities. The clients reproduce four WWW server traces. All these traces have been used in previous studies. Clarknet is a trace from a commercial Internet provider, Forth is from the FORTH Institute in Greece, Nasa is from NASA's Kennedy Space Center, and Rutgers contains the accesses made to the main server for the Computer Science Department at Rutgers University in the first 25 days of March 2000.

We eliminated all incomplete (due to network failures or client commands) requests in the traces and ended up with the characteristics listed in table 1. The traces cover a relatively wide spectrum of behavior. The number of files ranges from about 9100 to 28900 with an average file size between roughly 14 and 28 KBytes. The average

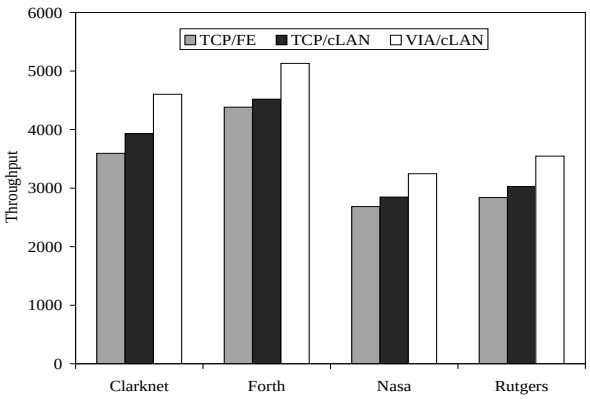

Figure 3: Throughput for protocol/network combinations.

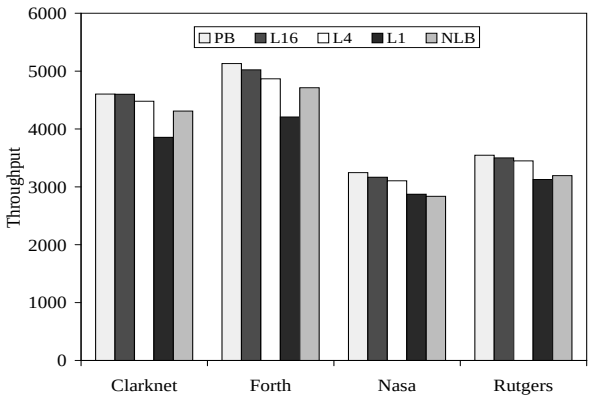

Figure 4: Throughput for different dissemination strategies.

size of the files serviced ranges from about 9 to 22 KBytes.

Metrics. We focus solely on the throughput (and messaging behavior, of course) of the different versions of our server, since server latencies are almost always low compared to the overall latency a client experiences establishing connections, issuing requests, and waiting for replies across a wide-area network. In order to determine the maximum throughput of each version of the server, we disregarded the timing information in the traces and made clients issue new requests as soon as possible. Before starting our measurements, we warm the node caches executing our traces for 5 minutes.

3.2 Effect of Overhead and Bandwidth

In this section we present the performance results of PRESS running on our cluster with three protocol/network combinations. For all these combinations the communication with the clients is through TCP over the Fast Ethernet network. The combinations are:

- TCP/FE. Intra-cluster communication uses TCP through additional Fast Ethernet interfaces. Under this protocol/network combination, sending a 4-byte message takes 82 microseconds, whereas the observed bandwidth achieved for 32-KByte messages is 11.5 MBytes/s.
- TCP/cLAN. The server still uses TCP for all the intra-cluster communication, but the cLAN is used for this communication. This implementation runs the complete TCP stack, just like TCP/FE; it does not take advantage of the reliability and message ordering properties of the cLAN. Under this combination, sending a 4-byte message takes 76 microseconds and the observed bandwidth is 32 MBytes/s with 32-KByte messages.
- VIA/cLAN. The server uses VIA over cLAN for all the intra-cluster communication. Under this combination, sending a 4-byte message takes 9 microseconds. The network peaks at 102 MBytes/s bandwidth for 32-KByte messages.

Figure 3 plots the throughput of PRESS for our four traces under the three different protocol/network combinations. By comparing TCP/FE and TCP/cLAN we can isolate the effect of network bandwidth on performance, since cLAN bandwidth is a factor of 3 higher than that of Fast Ethernet and overheads are virtually the same. This comparison shows that increased network bandwidth provides relatively small performance benefits, even for the traces that involve large requested files (Rutgers and Nasa). The average difference in performance between TCP/FE and TCP/cLAN over all traces is 6%.

The comparison between TCP/cLAN and VIA/cLAN isolates the effect of processor overhead on performance, since the VIA overhead is a factor of 8 lower than that of TCP and network bandwidth is not a serious performance factor as we just saw. This comparison shows that the gains that can be accrued by exploiting a protocol with lower processor overhead are more significant, ranging from 14% for Forth to 17% for Rutgers.

3.3 Effect of Strategies for Load Information Dissemination

The results presented in the previous section correspond to the version of PRESS that piggy-backs load information in the messages exchanged among the nodes. As the load at a node can change frequently, this piggy-backing strategy may leave other nodes with an inconsistent view of the amount of load at the node. Such an inconsistent view may cause PRESS to make poor load balancing decisions.

Based on this observation, in this subsection we evaluate the cost of utilizing a more aggressive strategy for distributing load information in the VIA/cLAN implementation. More specifically, we consider a strategy that directs a node to broadcast its current load when the load value is a certain number of connections (load threshold) greater or smaller than the last broadcast value. In effect, we study the tradeoff between the level of consistency of the load information and the overhead of the extra messages. Given that we rely on user-level communication over a high-bandwidth network for these experiments, this tradeoff may be favorable to exchanging more messages. As we see next, this is *not* the case.

Figure 4 presents the throughput of PRESS for five different load information dissemination strategies. The rightmost bar for each trace represents the base line case: no load balancing at all (labeled "NLB"). The next three bars from right to left represent load thresholds of 1 ("L1"), 4 ("L4"), and 16 ("L16") connections. The leftmost bar represents the throughput of the default, piggy-backing implementation ("PB"). The number of messages, bytes, and the average size of messages involved in each of these implementations is presented in table 2.

We can make several observations from this figure. The first

Version	Msg type	Num msgs (K)	Num bytes (MB)	Avg msg size
NLB	Load	0.0	0.0	0.0
	Flow	1203.6	15.3	13.0
	Forward	2075.5	107.1	52.9
	Caching	39.0	2.2	58.9
	File	2699.8	19479.4	7388.4
	TOTAL	6017.8	19604.0	–
L1	Load	29902.2	458.8	15.7
	Flow	8279.0	105.1	13.0
	Forward	1605.9	82.9	52.9
	Caching	52.5	3.0	58.8
	File	2092.4	15052.3	7366.4
	TOTAL	41932.0	15702.2	–
L4	Load	6176.6	96.4	16.0
	Flow	2636.2	33.5	13.0
	Forward	1879.1	97.0	52.9
	Caching	48.6	2.8	58.8
	File	2445.1	17687.3	7407.4
	TOTAL	13185.6	17917.0	–
L16	Load	342.2	5.3	16.0
	Flow	1155.6	14.7	13.0
	Forward	1839.2	94.9	52.9
	Caching	51.5	3.0	58.8
	File	2389.4	17194.8	7369.1
	TOTAL	5777.8	17312.8	–
PB	Load	0.0	0.0	0.0
	Flow	1152.4	19.1	17.0
	Forward	1984.6	110.2	56.8
	Caching	48.1	3.0	62.8
	File	2576.8	18580.5	7383.7
	TOTAL	5761.9	18712.7	–

Table 2: Intra-cluster communication and dissemination strategies.

observation is that avoiding load information broadcasts is always the best approach, even under such favorable communication conditions. In fact, broadcasting with a load threshold of 1 connection can perform worse than not doing load balancing within the cluster at all, especially for the traces that exhibit higher throughput (i.e. faster load variations). Increasing the threshold reduces the number of messages tremendously (as shown in table 2) and increases overall performance. However, the best implementation is indeed the one that uses piggy-backing; it combines the minimum number of messages with good enough load balancing.

As one would expect, using remote memory writes for the load broadcasts improves the performance of L1 significantly, improves the performance of L4 slightly, and does not affect L16. Even when using remote memory writes for load broadcasts, the piggy-backing version is at least as efficient as any other version of PRESS.

3.4 Effect of Remote Writes and Zero-Copy

To evaluate the usefulness of remote memory writes and zero-copy transfers, we developed 5 other versions of PRESS, numbered from 1 to 5. The version we studied so far, labeled "VIA/cLAN" in figure 3 and "PB" in figure 4, is called version 0. The versions differ in the extent to which remote writes and zero-copy are utilized. In all versions, load information is piggy-backed in other messages and intra-cluster communication uses VIA over cLAN. A summary with the description of each version of PRESS appears in table 3.

	Versions					
Message	V0	V1	V2	V3	V4	V5
Flow	reg	rmw	rmw	rmw	rmw	rmw
Forward	reg	reg	rmw	rmw	rmw	rmw
Caching	reg	reg	rmw	rmw	rmw	rmw
File	reg	reg	reg	rmw	rmw + 0-cp RX	rmw + 0-cp TX and RX

Table 3: Communication characteristics of PRESS versions. Keys: reg = regular, rmw = remote memory writes, 0-cp = zero-copy, TX = send, RX = receive.

Version 0 only utilizes regular messages, i.e. a receiver is interrupted to handle an arriving message and data copies are made at the sender and at the receiver for all message types. Note that most of the data copies in version 0 are of no serious performance consequence (very little data are involved in each case), except for the copies made at the sender and receiver of a file transfer message. The copy at the receiver side is unavoidable when using regular messages, since files can be bigger than message descriptors and the corresponding descriptor must be freed as quickly as possible for use by another message. The copy at the sender side could be avoided, provided that we could either: make all cache pages unswappable and available to VIA, or manage a smaller cache of unswappable and available pages with low overhead.

Version 1 improves on version 0 by using remote memory writes for flow control messages. In this type of messages only a word of data is transferred per message. The messages do not require immediate attention and can be overwritten.

Version 2 improves on version 1 by using remote memory writes for forward and caching information messages, as well as flow-control messages. Forward and caching messages are very similar in that a file name is transferred in each message, the messages can not be overwritten, and polling fairly frequently for these messages can improve performance. At each node, two circular buffers are allocated for forward and caching messages for each other node. As each node knows the location of its private buffers at every other node, it keeps track of exactly where the next file name should be written in the memories of remote nodes. Polling is done by looking at message sequence numbers stored at the last position of each (fixed-size) buffer entry. A receiver determines that the next message has not yet been completely received if its sequence number is not equal to the last sequence number plus 1. Processors poll for these messages at the end of the main server loop.

Version 3 improves on version 2 by adding remote memory writes for file transfers. The data structures for these transfers are a small circular buffer that is similar to the forward and caching buffers just described and a large circular buffer for the actual file data. These two buffers are allocated by each node for each other node. Again, because each node knows the location of its private buffers at every other node, it keeps track of exactly where to write the memories of remote nodes. The sender of a file then writes the file data to the large buffer and the file information to the small buffer. The file informa-

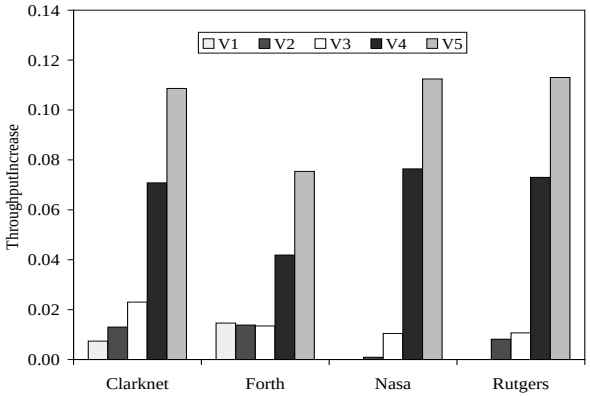

Figure 5: Throughput for the RMW and zero-copy versions.

tion includes a pointer to the beginning of the file in the large buffer. Again, polling by the receiver is done on the sequence numbers at the end of the server loop. When the receiver detects the arrival of a file, it copies it to another buffer and sends it back to the requesting client. This version does not require a receive thread, as all communication is performed with remote memory writes.

Version 4 improves on version 3 by having the receiver of file data send the data to the client right out of the large communication buffer. By doing this, version 4 eliminates the expensive data copy made at the receiver of a file transfer message.

Finally, version 5 improves on version 4 by eliminating the data copy at the transmitter of a file transfer message. For that, all the pages corresponding to cached files are registered in VIA.

Figure 5 depicts the throughput increase of the different versions of PRESS for each trace on our 8-node cluster server. The throughput increases are determined with respect to version 0. Table 4 lists the number of messages and bytes that each version transfers. Results for version 0 are labeled "PB" in table 2.

The figure shows that versions 1 and 2 produce minimal improvements in comparison to version 0. This is explained by the relatively small number of messages of these types. Version 3 does not significantly improve performance either, even though interrupts for message reception and the receive thread itself have been eliminated. The reason for this surprising result is that using remote memory writes requires two messages per file (one with the file data and one

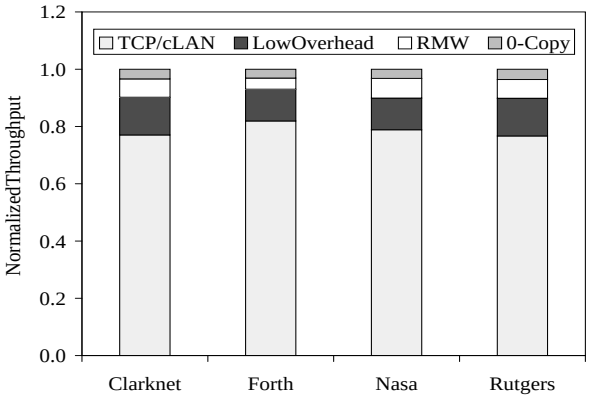

Figure 6: Summary of contributions.

Version	Msg type	Num msgs (K)	Num bytes (MB)	Avg msg size
V1	Flow	1161.6	4.5	4.0
	Forward	2001.5	111.1	56.8
	Caching	51.0	3.1	62.8
	File	2594.0	18610.7	7346.6
	TOTAL	5808.2	18729.4	–
V2	Flow	1856.7	7.3	4.0
	Forward	1980.1	102.2	52.8
	Caching	51.6	2.8	54.8
	File	2567.4	18452.2	7359.7
	TOTAL	6455.8	18564.5	–
V3	Flow	4194.2	65.5	16.0
	Forward	2026.2	120.4	60.8
	Caching	51.1	3.1	62.8
	File	4644.4	18818.3	4149.1
	TOTAL	10915.9	19007.3	–
V4	Flow	4603.4	70.5	15.7
	Forward	2122.3	126.1	60.9
	Caching	47.2	2.9	62.8
	File	4870.2	19827.9	4169.0
	TOTAL	11643.1	20027.5	–
V5	Flow	5176.5	75.2	14.9
	Forward	2230.7	132.5	60.8
	Caching	47.4	2.9	62.8
	File	5117.7	20754.5	4152.7
	TOTAL	12572.4	20965.1	–

Table 4: Intra-cluster communication, RMW, and zero-copy.

with metadata), rather than a single message. The effect of this increase can be observed by comparing the number of file messages and the total number of messages in versions 2 and 3 in table 4.

In contrast, versions 4 and 5 do exhibit non-trivial performance benefits. The benefits from version 4 range from 4% for Forth to 8% for Nasa, averaging 6.6%. Version 5 outperforms all other versions, reaching improvements that range from 8% for Forth to 11% for Rutgers. The gains achieved by these two versions stem from avoiding large data copies. *Note however that remote memory writes have commonly been studied in the literature in scenarios where they obviate the need for data copies at receivers (e.g. [28]), so we credit the gains achieved by version 4 to remote memory writes and the gains of version 5 to zero-copy transfers.*

Overall, by comparing version 5 against the TCP/cLAN implementation, we find that user-level communication improves throughput by as much as 29%, averaging 26%. No specific property of this type of communication is responsible for the vast majority of the gains. Low processor overhead is responsible for about 15%, remote memory writes for file transfers are responsible for about 7%, and zero-copy file transfers are responsible for about 4%. A summary of these results is presented in figure 6. The throughput gains provided by low overhead, remote memory writes, and zero-copy transfers are stacked above the base TCP/cLAN throughput.

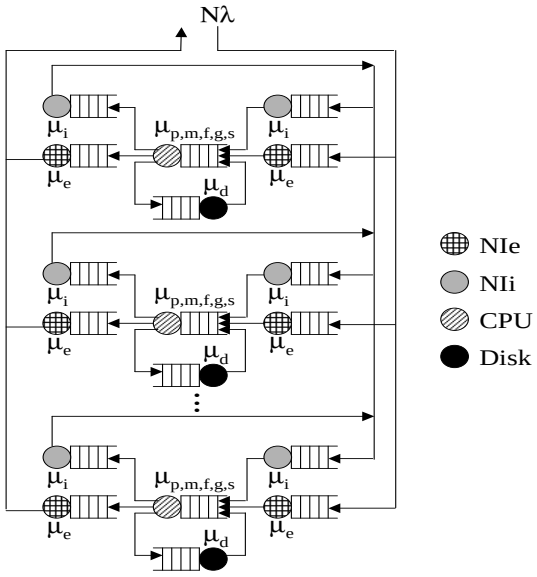

Figure 7: Model of a locality-conscious server.

4 Modeling

In this section we analytically extrapolate our experimental results to a broader range of parameters. Our goal is to determine the benefits that can be achieved by low processor overhead, remote memory writes, and zero-copy transfers under a variety of current and future scenarios. We now turn to a description of the model.

4.1 The Model

Figure 7 depicts our open queueing model of a portable locality-conscious server such as PRESS running on a cluster of N PCs with an intra-cluster or "internal" network and an "external" network that connects the cluster to the outside world. The model mimics our own cluster.

The model assumes that all queues are M/M/1. In the model, requests for files arrive at a rate $N \times \lambda$. Assuming that the traffic to the nodes is perfectly balanced, the probability that a request is assigned to a specific workstation is the same for all the workstations ($1/N$). Thus, each external network interface receives requests with a rate λ and processes them at a rate μ_e. The initial processing of requests (reading and parsing) is done by the CPU at a rate μ_p. If the requested file is cached locally the node just replies to the request at a rate μ_m. If the file is cached only remotely, the request is forwarded by the CPU at a rate μ_f. Each internal interface processes forwarded requests and replies at a rate μ_i.

After a request is received by the service node, the server tests whether the requested file is cached and, if so, the file is sent back by the CPU to the initial node through the internal cluster network with a rate μ_s. Otherwise, the node must first read the file from disk, store it in its memory, and then transfer the file across the cluster (again with a rate μ_s). Each disk receives read requests with a rate $(1 - H) \times \lambda$ (where H is the probability that a requested file is cached in main memory) and processes them at a rate μ_d. The CPU of the initial node receives the reply of a forwarded request at a rate μ_g. Finally, when the requested file is ready to be sent out, it is sent at a rate μ_e.

Other important parameters to our model are the cache size per node (C), the average size of the requested files (S), and the file request distribution. This latter parameter deserves further comments. In this study, we concentrate on heavy-tailed distributions of access, such as the ones exhibited by WWW servers [9]. Such distributions can be approximated by means of Zipf-like distributions [9], where the probability of a request for the i'th most popular file is proportional to $1/i^{\alpha}$ with α typically taking on some value less than unity.

With these parameters, we define the total cache space and the average cache hit rate for a locality-conscious server. The cache space is $C_{lc} = N \times C$ bytes if no file replication is allowed, or $C_{lc} = N \times (1 - R) \times C + R \times C$ bytes, if an R percentage of the main memory is used for file replication. The average cache hit rate can be defined as: $H = z(n, F)$, where $z(n, F)$ represents the accumulated probability of requesting the n most accessed files in a Zipf-like distribution of the requests to F files. The number of cached files (n) is equal to $\min(C_{lc} \div S, F)$. Furthermore, the percentage of requests forwarded to another workstation can be defined as: $Q = (N - 1) \times (1 - h) \div N$, where the hit rate for replicated files (h) is equal to $z(\min(R \times C \div S, F), F)$. Note that our use of $z(n, F)$ to describe hit rates effectively means that the most accessed files are always cached in the model.

To simplify the presentation, we define the locality-conscious hit rate (H_{lc}) as a function of a single-node server's hit rate (H_{sn}) as follows: $H_{lc} = z(\min(C_{lc} \div S, f), f)$, where f is such that $H_{sn} = z(C \div S, f)$. In the same way, we define the percentage of requests forwarded to another node as: $Q = (N - 1) \times (1 - h) \div N$, where $h = z(R \times C \div S, f)$.

Note that all the above definitions assume that the probability of an unreplicated file to be cached by a specific workstation is the same for all the workstations ($1/N$). The first two columns of table 5 summarize the model parameters and their descriptions.

As our model assumes a cost-free distribution algorithm, cost-free caching information dissemination and message flow control, perfect load balancing, and does not consider cache replacements and contention for network wires and memory and I/O buses, it provides an *upper bound on the throughput* achievable by these servers.

Parameter Values. We carefully selected default values for our model parameters. The value of R was chosen to maximize the performance of the servers. To concentrate solely on processor overhead, remote memory writes, and zero-copy, we assume peak bandwidths for the internal and external networks. The service rate of the external network interface, μ_e, was selected assuming that the interface provides 100 Mbits/s full-duplex links with 4-microsecond overhead at the network interface per message. These parameters approximate the characteristics of our Fast Ethernet network interface cards. The service rate of the internal network interface, μ_i, was selected assuming that the interface provides 1 Gbit/s full-duplex links with 3-microsecond overhead at the network interface per message. These parameters approximate the characteristics of our Giganet interface cards. The fixed cost in the μ_m, μ_s, and μ_g rates are based on measurements we took by running a single request at a time through our own real server on two cluster nodes. The μ_d and μ_p rates are based on these same measurements.

Param	Description	Definition or Default Value
R	Percentage of replication	15%
α	Zipf constant	0.8
μ_i	NIi transfer rate	(0.000003 + $size$/125000)$^{-1}$ ops/s
μ_e	NIe transfer rate	(0.000004 + $size$/125000)$^{-1}$ ops/s
μ_p	Request read/parsing rate by CPU	5882 ops/s
μ_m	Client reply send rate (after stored locally) by CPU	(0.00027 + S/12500)$^{-1}$ ops/s
μ_d	Disk access rate	(0.0188 + S/3000)$^{-1}$ ops/s
μ_f	Intra-cluster request forwarding rate by CPU	31250 ops/s – VIA 3676 ops/s – TCP/cLAN
μ_s	Intra-cluster reply send rate by CPU	(0.00003 + S/125000)$^{-1}$ ops/s – VIA (0.00027 + S/125000)$^{-1}$ ops/s – TCP/cLAN
μ_g	Intra-cluster reply reception rate by CPU	(0.00003 + S/125000)$^{-1}$ ops/s – VIA (0.00027 + S/125000)$^{-1}$ ops/s – TCP/cLAN
C	Total cache space	C = 128 MBytes $C_{lc} = N \times (1 - R) \times C + R \times C$
H	Cache hit rate	$H_{sn} = z(\min(C \div S, F), F)$ $H_{lc} = z(\min(C_{lc} \div S, F), F)$
h	Cache hit rate for replicated files	$h = z(\min(R \times C \div S, F), F)$
Q	Percentage of requests forwarded	$Q = (N - 1) \times (1 - h) \div N$

Table 5: Model parameters and their default values. S = avg file size in KBytes; $size$ = avg transfer size in KBytes. F = number of files.

4.2 Results

To collect predictions from the model, we instantiate its parameters and solve its traffic equations using standard linear algebra techniques. To validate the model, we compared the throughputs of version 5 and TCP/cLAN on 8 nodes against the corresponding model predictions. These comparisons show that the version 5 performance is within 2% of the model results for traces with large average file sizes (Nasa and Rutgers). For traces with small average file sizes (Clarknet and Forth), version 5 is within 20% of the model results. The TCP/cLAN performance is within 15% and 25% of the model results for large and small average file sizes, respectively. These validation results demonstrate that the model provides a looser upper bound for traces with small average file sizes than with large average file sizes. On average, modeling and experimental results are within 14% of each other.

Effect of processor overhead. We start by studying the benefits achievable by lowering processor overheads for a wide range of workload working set sizes and numbers of cluster nodes. To simplify the analysis, we model variations in working set size using the cache hit rate of a single-node server; as the working set size increases, the cache hit rate decreases.

Figure 8 plots the throughput improvements that can be accrued by running a portable locality-conscious server with VIA-based intra-cluster communication in comparison to a similar server that uses TCP for the same purpose. The internal network bandwidth we assume, 128 MBytes/s, is the same for both systems, as listed in table 5. We also assume an average file size of 16 KBytes. The figure shows that for small numbers of nodes and very low hit rates, the throughput does not improve as a result of the lower processor overhead. The reason is that the disks are the performance bottleneck in this area of the parameter space. As the hit rate or the number of nodes is increased, the amount of data that can be cached also increases, and we start to see throughput improvements. For a fixed hit rate, increasing the number of nodes leads to significant throughput improvements at first, but quickly improvements level off. The reason for this effect is that intra-cluster communication increases by a factor of $1/(N \times (N - 1))$ for each node that is added to the system, where N is the size of the current configuration. For small N, increases in intra-cluster traffic are significant; for large N, they approach 0. Overall, improvements are most significant, 37%, for 128 nodes and 36% hit rate.

We now study throughput improvements due to lowering processor overheads as a function of average file sizes and numbers of nodes. Our real traces exhibit average file sizes of 8.8 to 21.8 KBytes and we expect average file sizes to continue growing. In figure 9, we investigate average file sizes of up to 128 KBytes, assuming a 90% single-node hit rate. The figure shows that improvements are most significant, 48%, for 4-KByte files and large numbers of nodes. As we increase the average file sizes, throughput improvements decrease significantly to about 4%. The reason for this result is that the overhead quickly becomes a small fraction of the overall cost of transferring messages.

These results suggest that lowering processor overheads can provide higher benefits than we found experimentally, but only for large working set sizes compared to the size of a single memory. Increases in average file sizes actually reduce the achievable improvements. Real life file sizes tend to continuously increase, which points to decreasing benefits. Depending on whether real life working sets grow faster than memory sizes, we may see greater benefits from low processor overhead, but they will likely not exceed 48%.

Effect of remote memory writes and zero-copy transfers. We now turn to studying the benefits achievable by exploiting remote memory writes and zero-copy transfers, again as a function of a range of single-node hit rates and numbers of nodes. Figure 10 plots the throughput improvements that can be accrued by exploiting these mechanisms in VIA with respect to a server that only uses regular, 1-copy VIA messages. The performance parameters are those listed in table 5. Again, we assume a 16-KByte file size. The figure shows the same overall trends as in figure 8. The main difference is that the

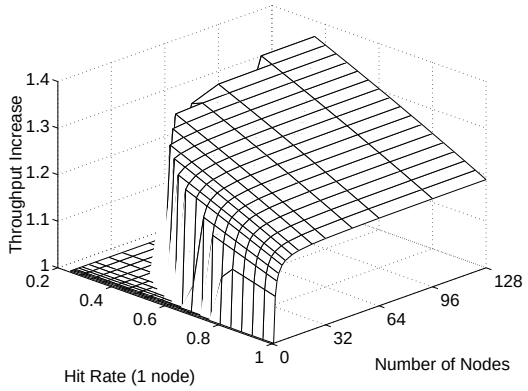

Figure 8: Gains achievable by lowering overheads, as a function of hit rate and number of nodes.

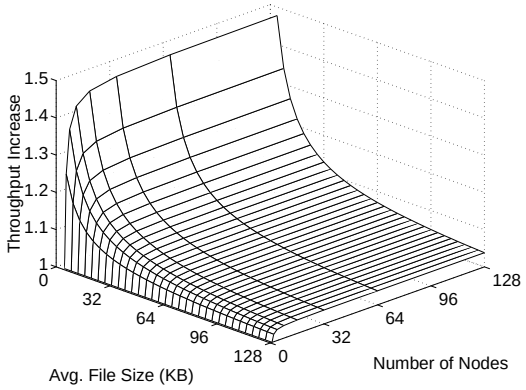

Figure 9: Gains achievable by lowering overheads, as a function of average file size and number of nodes.

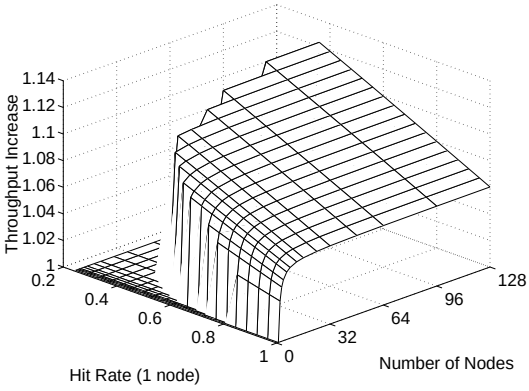

Figure 10: Gains achievable by using RMW and 0-copy, as a function of hit rate and number of nodes.

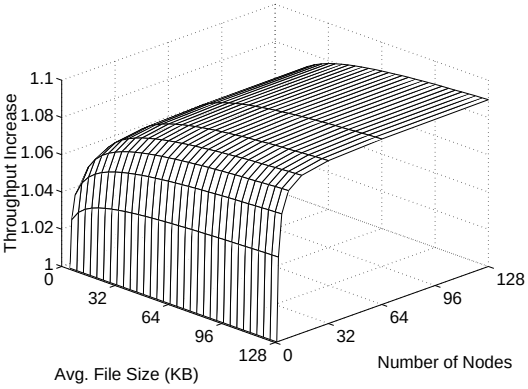

Figure 11: Gains achievable by using RMW and 0-copy, as a function of average file size and number of nodes.

maximum gain here is only 12%.

Figure 11 plots the improvements due to remote memory writes and zero-copy transfers, as a function of average file sizes and numbers of nodes. We again assume a 90% hit rate on a single node. For small file sizes, most of the benefit comes from avoiding interrupts with remote memory writes. As we increase the size of files, the throughput gains increase, now mostly as a result of zero-copy, and eventually reach 9%. It is somewhat surprising that the benefit of zero-copy does not grow almost linearly with the file size. This is because, as files become larger, the CPU spends longer to send them back to clients, continuously diminishing the fraction of time spent on intra-cluster communication. Varying the number of nodes has the same effect as we observed above.

These results suggest that exploiting remote memory writes and zero-copy transfers can provide slightly higher benefits than we found experimentally, but only for large working set sizes. Increases in average file sizes do not significantly increase the achievable improvements. Depending on whether real life working sets grow faster than memory sizes, we may see greater benefits from these mechanisms, but it is unlikely that they will exceed 12%.

Future systems. The above results show that in the best of circumstances user-level communication can increase throughputs by 49% (37% coming from low processor overhead as shown in figure 8 and 12% coming from remote memory writes and zero-copy transfers as shown in figure 10), assuming the characteristics of our cluster. It is important also to consider higher performance systems. In particular, we now consider the effect of faster processors and higher performance TCP implementations.

Faster processors would not increase the throughput improvements we have found. The reason for this is simply that gains from user-level communication are most significant when the processor is the bottleneck. Thus, increasing the speed of the processor scales all the relevant parameters by the same factor, keeping throughput improvements the same. This effect has been recognized before in [20].

Higher performance TCP communication *can* have an impact on throughput improvements. In particular, higher performance communication can be achieved with a higher bandwidth network and a zero-copy TCP implementation along the lines of the IO-lite operating system [26]. The idea is to use the cached file data, which is constantly pinned to memory, as a large TCP buffer. File data can then be sent to clients or to other nodes without copying them out of the cache. We expect next-generation operating systems to include such zero-copy optimizations.

We modeled such a system by halving the μ_m parameter and halving the fixed cost of the TCP versions of the μ_f, μ_s, and μ_g parameters of table 5. The results are shown in figure 12, assuming 16-KByte files, and figure 13, assuming 90% hit rate. The figures show that, under the best of circumstances, the throughput improvement

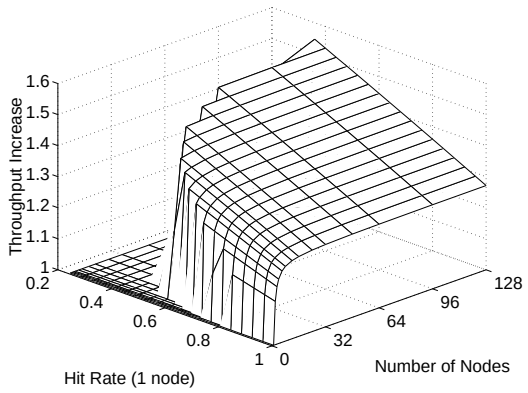

Figure 12: Gains achievable by user-level communication, as a function of hit rate and number of nodes.

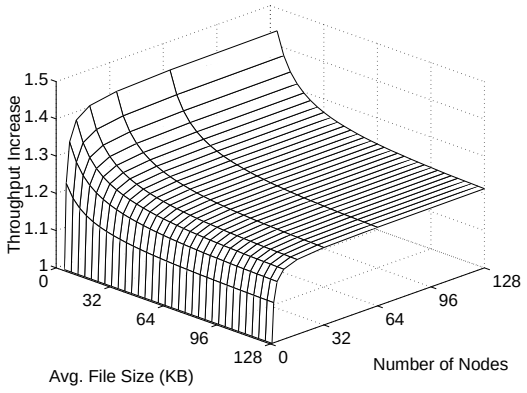

Figure 13: Gains achievable by user-level communication, as a function of average file size and number of nodes.

provided by user-level communication can reach 55%.

5 Conclusions

In this paper we have quantified the impact of user-level communication and network bandwidth on the performance of a content-aware server. Our results demonstrated that processor overhead, remote memory writes, and zero-copy can all provide performance gains, whereas network bandwidth is not as important.

Based on our results and experience, we conclude that low overhead, remote memory writes, and zero-copy transfers should continue to be provided and exploited in future user-level communication systems. In addition, we conclude that user-level communication can improve the performance of content-aware servers by as much as 49% for current operating systems and 55% for next-generation operating systems. Whether such high gains will ever be achieved depends on working sets growing faster than memories.

References

[1] S. Araki, A. Bilas, C. Dubnicki, J. Edler, K. Konishi, and J. Philbin. User-Space Communication: A Quantitative Study. In *Proceedings of Supercomputing '98*, November 1998.

[2] M. Aron, P. Druschel, and W. Zwaenepoel. Efficient Support for P-HTTP in Cluster-Based Web Servers. In *Proceedings of the USENIX 99 Annual Technical Conference*, Monterey, CA, June 1999.

[3] M. Aron, D. Sanders, P. Druschel, and W. Zwaenepoel. Scalable Content-Aware Request Distribution in Cluster-Based Network Servers. In *Proceedings of USENIX'2000 Technical Conference*, San Diego, CA, June 2000.

[4] M. Banikazemi, V. Moorthy, L. Herger, D. K. Panda, and B. Abali. Efficient Virtual Interface Architecture Support for the IBM SP Switch-Connected NT Clusters. In *Proceedings of the 14th International Parallel and Distributed Processing Symposium*, Cancun, Mexico, May 2000.

[5] R. Bianchini and E. V. Carrera. Analytical and Experimental Evaluation of Cluster-Based WWW Servers. *World Wide Web Journal*, 3(4):215–229, December 2000.

[6] M. Blumrich, K. Li, R. Alpert, C. Dubnicki, E. Felten, and J. Sandberg. Virtual Memory Mapped Network Interface for the SHRIMP Multicomputer. In *Proceedings of the 21st Annual International Symposium on Computer Architecture*, pages 142–153, Chicago, IL, April 1994.

[7] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W.-K. Su. Myrinet: A Gigabit per Second Local Area Network. *IEEE Micro*, 15(1):29–36, February 1995.

[8] P. Bozeman and B. Saphir. A Modular High Performance Implementation of the Virtual Interface Architecture. In *Proceedings of the 2nd Extreme Linux Workshop*, Monterey, CA, June 1999.

[9] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web Caching and Zipf-like Distributions: Evidence and Implications. In *Proceedings of IEEE INFOCOM 99*, pages 126–134, New York, NY, March 1999.

[10] P. Buonadonna, A. Geweke, and D. Culler. An Implementation and Analysis of the Virtual Interface Architecture. In *Proceedings of Supercomputing'98*, Orlando, FL, November 1998.

[11] E. V. Carrera and R. Bianchini. Evaluating Cluster-Based Network Servers. In *Proceedings of the 9th IEEE International Symposium on High Performance Distributed Computing*, pages 63–70, Pittsburgh, PA, August 2000.

[12] E. V. Carrera and R. Bianchini. Efficiency vs. Portability in Cluster-Based Network Servers. In *Proceedings of the 8th Symposium on Principles and Practice of Parallel Programming*, Snowbird, UT, June 2001.

[13] Compaq Corp., Intel Corp., and Microsoft Corp. *Virtual Interface Architecture Specification, Version 1.0*, 1997.

[14] F. M. Cuenca-Acuna and T. D. Nguyen. Cooperative Caching Middleware for Cluster-Based Servers. In *Proceedings of the 10th International Symposium on High Performance Distributed Computing*, San Francisco, CA, August 2001.

[15] D. Dunning and G. Regnier and G. McAlpine and D. Cameron and B. Shubert and F. Berry and A. M. Merritt and E. Gronke and C. Dodd. The Virtual Interface Architecture. *IEEE Micro*, 18(2):66–76, March 1998.

[16] R. dos Santos, R. Bianchini, and C. L. Amorim. A Survey of Messaging Software Issues and Systems for Myrinet-Based Clusters. *Parallel and Distributed Computing Practices, special issue on High-Performance Computing on Clusters*, June 1999.

[17] C. Dubnicki, L. Iftode, E. Felten, and K. Li. Software Support for Virtual Memory-Mapped Communication. In *Proceedings of the 10th International Parallel Processing Symposium*, April 1996.

[18] Giganet. *Giganet cLAN Cluster Switch*. http://www.giganet.com/.

[19] S Gopalakrishnan *et al.* The Federated File System. Technical report, Department of Computer Science, Rutgers University, 2000. In preparation.

[20] T. Heath, S. Kaur, R. Martin, and T. Nguyen. Quantifying the Impact of Architectural Scaling on Communication. In *Proceedings of the 7th IEEE Symposium on High-Performance Computer Architecture*, January 2001.

[21] V. Holmedahl, B. Smith, and T. Yang. Cooperative Caching of Dynamic Content on a Distributed Web Server. In *Proceedings of the 7th IEEE International Symposium on High Performance Distributed Computing*, pages 243–250, July 1998.

[22] R. Martin and D. Culler. NFS Sensitivity to High Performance Networks. In *Proceedings of the International Conference on Measurement and Modeling of Computer Systems*, April 1999.

[23] R. Martin, A. Vahdat, D. Culler, and T. Anderson. Effects of Communication Latency, Overhead, and Bandwidth in a Cluster Architecture. In *Proceedings of the 24th International Symposium on Computer Architecture*, June 1997.

[24] V. S. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, and E. Nahum. Locality-Aware Request Distribution in Cluster-based Network Servers. In *Proceedings of the 8th ACM Conference on Architectural Support for Programming Languages and Operating Systems*, pages 205–216, San Jose, CA, October 1998.

[25] V. S. Pai, P. Druschel, and W. Zwaenepoel. Flash: An Efficient and Portable Web Server. In *Proceedings of the USENIX 99 Annual Technical Conference*, June 1999.

[26] V. S. Pai, P. Druschel, and W. Zwaenepoel. IO-Lite: A Unified I/O Buffering and Caching System. *ACM Transactions on Computer Systems*, 18(1):37–66, 2000.

[27] S. Pakin, M. Karamcheti, and A. Chien. Fast Messages (FM): Efficient, Portable Communication for Workstation Clusters and Massively-Parallel Processors. *IEEE Parallel and Distributed Technology*, 5(2):60–73, April 1997.

[28] M. Rangarajan and L. Iftode. Software Distributed Shared Memory over Virtual Interface Architecture: Implementation and Performance. In *Proceedings of the 3rd Extreme Linux Workshop*, October 2000.

[29] Resonate. Resonate Central Dispatch. http://www.resonateinc.com/.

[30] Y. Saito, B. N. Bershad, and H. M. Levy. Manageability, Availability and Performance in Porcupine: A Highly Scalable, Cluster-Based Mail Service. In *Proceedings of 17th ACM Symposium on Operating Systems Principles*, December 1999.

[31] ServerIron. ServerIron. http://www.foundrynetworks.com/products/Webswitches.html.

[32] E. Speight, H. Abdel-Shafi, and J. K. Bennett. Realizing the Performance Potential of the Virtual Interface Architecture. In *Proceedings of the International Conference on Supercomputing*, pages 184–192, Rhodes, Greece, June 1999.

[33] R. Stets, S. Dwarkadas, L. Kontothanassis, U. Rencuzogullari, and M. Scott. The Effect of Network Total Order, Broadcast, and Remote-Write Capability on Network-Based Shared Memory Computing. In *Proceedings of the 6th International Symposium on High-Performance Computer Architecture*, January 2000.

[34] H. Tesuka, A. Hori, and Y. Ishikawa. PM: A High-Performance Communication Library for Multi-User Parallel Environments. Technical Report TR-96015, Real World Computing Parnership, November 1996.

[35] T. von Eicken, A. Basu, V. Buch, and W. Vogels. U-Net: A User-Level Network Interface for Parallel and Distributed Computing. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles*, pages 40–53, Copper Mountain, Colorado, December 1995.

[36] K. G. Yocum, J. S. Chase, A. J. Gallatin, and A. R. Lebeck. Cut-Through Delivery in Trapeze: An Exercise in Low Latency Messaging. In *Proceedings of the IEEE Symposium on High-Performance Distributed Computing*, Portland, OR, August 1997.

[37] H. Zhu, B. Smith, and T. Yang. Scheduling Optimization for Resource-Intensive Web Requests on Server Clusters. In *Proceedings of the 11th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 13–22, June 1999.